COMP 110/L Lecture 16

Mahdi Ebrahimi

Some slides are adapted from Dr. Kyle Dewey

Outline

- Multidimensional arrays
- JUnit `fail()`

Multidimensional Arrays

Recap - Arrays

Arrays are fixed-length sequences
of elements of the same type.

Recap - Arrays

Arrays are fixed-length sequences of elements of the same type.

```
new char[]{'a', 'b', 'c'}

new int[]{1, 2, 3}

new String[]{"foo", "bar"}

new double[]{1.2, 3.4}
```

Motivations

Thus far, you have used one-dimensional arrays to model linear collections of elements. You can use a two-dimensional array to represent a matrix or a table. For example, the following table that describes the distances between the cities can be represented using a two-dimensional array.

Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0

Motivations

```
double[][] distances = {
   {0, 983, 787, 714, 1375, 967, 1087},
   {983, 0, 214, 1102, 1763, 1723, 1842},
   {787, 214, 0, 888, 1549, 1548, 1627},
   {714, 1102, 888, 0, 661, 781, 810},
   {1375, 1763, 1549, 661, 0, 1426, 1187},
   {967, 1723, 1548, 781, 1426, 0, 239},
   {1087, 1842, 1627, 810, 1187, 239, 0},
};
```

Multidimensional Arrays

Java also allows us to make arrays of *arrays*.

These are often called *multidimensional* arrays.

Multidimensional Arrays

Java also allows us to make arrays of *arrays*.
These are often called *multidimensional* arrays.

```
new int[][]{ new int[]{1, 2, 3},
             new int[]{4, 5},
             new int[]{6},
             new int[0],
             new int[]{7, 8, 9} }
```

Multidimensional Arrays

Java also allows us to make arrays of *arrays*.
These are often called *multidimensional* arrays.

```
new int[][]{ new int[]{1, 2, 3},
             new int[]{4, 5},
             new int[]{6},
             new int[0],
             new int[]{7, 8, 9} }
```

Corresponding type: `int[][]`

Multidimensional Array Utility

Commonly used for representing tables

Multidimensional Array Utility

Commonly used for representing tables

13	12	19
64	89	247
78	57	21

Multidimensional Array Utility

Commonly used for representing tables

13	12	19
64	89	247
78	57	21

```
new int[][]{ new int[]{13, 12, 19},
             new int[]{64, 89, 247},
             new int[]{78, 57, 21} }
```

Accessing Rows

One row of a two-dimensional array is an array...

Accessing Rows

One row of a two-dimensional array is an array...

```
int[][] array = ...;
int[] row = array[0];
```

Accessing Rows

One row of a two-dimensional array is an array...

```
int[][] array = ...;
int[] row = array[0];
```

Accessing Columns

...and columns are individual elements of rows.

Accessing Rows

One row of a two-dimensional array is an array...

```
int[][] array = ...;
int[] row = array[0];
```

Accessing Columns

...and columns are individual elements of rows.

```
int[][] array = ...;
int[] row = array[0];
int columnElement = row[5];
```

Accessing Rows

One row of a two-dimensional array is an array...

```
int[][] array = ...;
int[] row = array[0];
```

Accessing Columns

...and columns are individual elements of rows.

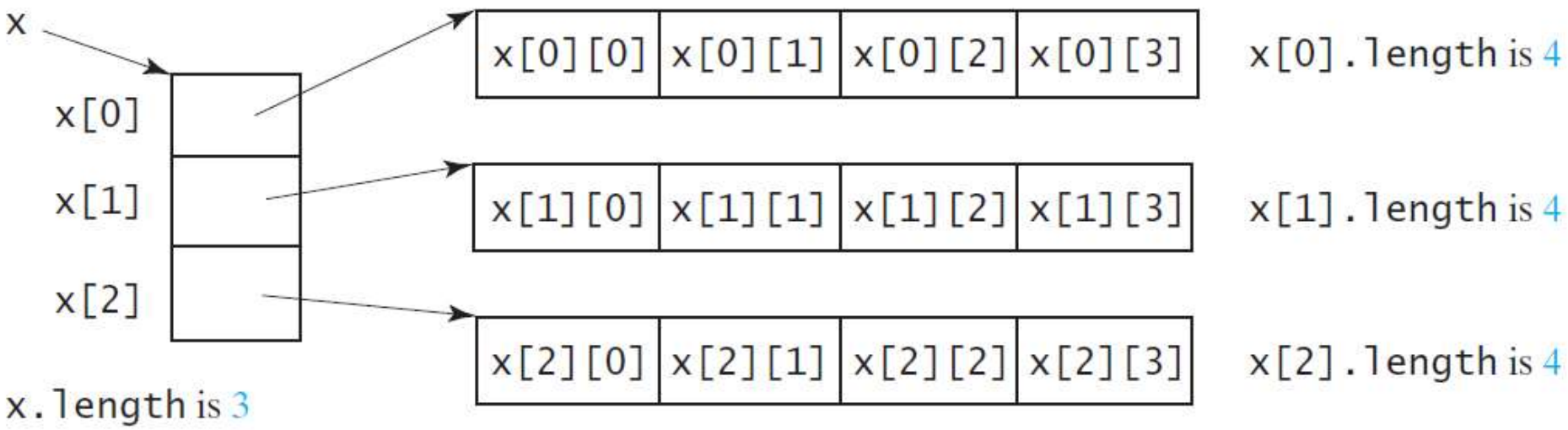
```
int[][] array = ...;
int[] row = array[0];
int columnElement = row[5];
```

```
int[][] array = ...;
int columnElement = array[0][5];
```

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Lengths of Two-dimensional Arrays

```
int[][] x = new int[3][4];
```



Lengths of Two-dimensional Arrays, cont.

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

array.length

array[0].length

array[1].length

array[2].length

array[3].length

array[4].length ArrayIndexOutOfBoundsException

Ragged Arrays

Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as *a ragged array*.

For example,

```
int[][] matrix = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```

```
matrix.length is 5
matrix[0].length is 5
matrix[1].length is 4
matrix[2].length is 3
matrix[3].length is 2
matrix[4].length is 1
```

Example:

`AccessTwoDimensionalElement.java`

More 2D Array Examples

- `PrintRow2D.java`
- `PrintCol2D.java`

JUnit fail()

fail()

Triggers immediate test failure

fail()

Triggers immediate test failure

```
Import static org.junit.Assert.fail;
```

fail()

Triggers immediate test failure

```
import static org.junit.Assert.fail;


    @Test
    public void testSomething() {
      if(someFailureCondition) {
        fail();
      }
    }
```

fail() Utility

- Some test failures cannot be easily phrased as one value equals another value
- Occasionally more convenient
- We can define our own `assertEquals()` and `assertArrayEquals()` using `fail()`

Some cases where it is useful:

1- mark a test that is incomplete, so it fails and warns you until you can finish it

2- making sure an exception is thrown:

There are three states that your test case can end up in
Passed: The function under test executed successfully and returned data as expected
Not Passed: The function under test executed successfully but the returned data was not as expected
Failed: The function did not execute successfully and this was not intended (Unlike negative test cases that expect a exception to occur).

If you are using eclipse there three states are indicated by a **Green**, **Blue** and **red** marker respectively.

We can use the fail operation for the third scenario.
e.g.:

public Integer add(integer a, Integer b) { return new Integer(a.intValue() + b.intValue())}

Passed Case: a = new Interger(1), b= new Integer(2) and the function returned 3
Not Passed Case: a = new Interger(1), b= new Integer(2) and the function returned any value other than 3
Failed Case: a = null , b = null and the function throws a NullPointerException

Example

- `FailExample.java`
- `FailExampleTest.java`